

RRRRRRRRRRRR		MMM		MMM	SSSSSSSSSSSS
RRRRRRRRRRRR		MMM		MMM	SSSSSSSSSSSS
RRRRRRRRRRRR		MMM		MMM	SSSSSSSSSSSS
RRR	RRR	MMMMMM	MMMMMM	SSS	
RRR	RRR	MMMMMM	MMMMMM	SSS	
RRR	RRR	MMMMMM	MMMMMM	SSS	
RRR	RRR	MMM	MMM	SSS	
RRR	RRR	MMM	MMM	SSS	
RRR	RRR	MMM	MMM	SSS	
RRRRRRRRRRRR		MMM		SSSSSSSSSS	
RRRRRRRRRRRR		MMM		SSSSSSSSSS	
RRRRRRRRRRRR		MMM		SSSSSSSSSS	
RRR	RRR	MMM			SSS
RRR	RRR	MMM			SSS
RRR	RRR	MMM			SSS
RRR	RRR	MMM			SSS
RRR	RRR	MMM			SSS
RRR	RRR	MMM			SSS
RRR	RRR	MMM			SSS
RRR	RRR	MMM		SSSSSSSSSSSS	
RRR	RRR	MMM		SSSSSSSSSSSS	
RRR	RRR	MMM		SSSSSSSSSSSS	

SY

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

PI


```
1 0001 0 MODULE RM3UPDATE (LANGUAGE (BLISS32) ,
2 0002 0 IDENT = 'V04-000'
3 0003 0 ) =
4 0004 1 BEGIN
5 0005 1
6 0006 1 *****
7 0007 1 *
8 0008 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY *
9 0009 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS. *
10 0010 1 * ALL RIGHTS RESERVED. *
11 0011 1 *
12 0012 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED *
13 0013 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE *
14 0014 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER *
15 0015 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY *
16 0016 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY *
17 0017 1 * TRANSFERRED. *
18 0018 1 *
19 0019 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE *
20 0020 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT *
21 0021 1 * CORPORATION. *
22 0022 1 *
23 0023 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS *
24 0024 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL. *
25 0025 1 *
26 0026 1 *****
27 0027 1
28 0028 1
29 0029 1 ++
30 0030 1
31 0031 1 FACILITY: RMS32 INDEX SEQUENTIAL FILE ORGANIZATION
32 0032 1
33 0033 1 ABSTRACT:
34 0034 1 High level update and other miscellaneous update specific
35 0035 1 routines.
36 0036 1
37 0037 1
38 0038 1 ENVIRONMENT:
39 0039 1
40 0040 1 VAX/VMS OPERATING SYSTEM
41 0041 1
42 0042 1 --
43 0043 1
44 0044 1
45 0045 1 AUTHOR: Todd M. Katz RE-CREATION DATE: 22-Dec-1982
46 0046 1
47 0047 1
48 0048 1 MODIFIED BY:
49 0049 1
50 0050 1 V03-026 JWT0184 Jim Teague 21-Jun-1984
51 0051 1 Fix invalid RRV corruption problem. On an update,
52 0052 1 if the new record is longer than the old record,
53 0053 1 a delete + insert operation is performed. However,
54 0054 1 one thing that is different about the insert is
55 0055 1 that the new record has to have the same record
56 0056 1 id as the old one. RMS saves the record id
57 0057 1 very early on in the update process, and never
```


58	0058	1	looks at it again before stuffing it into the new
59	0059	1	record. If the desired record has been displaced
60	0060	1	by a bucket split since last_id has been saved,
61	0061	1	the record id of the new record will almost
62	0062	1	certainly be wrong. To fix this, reevaluate
63	0063	1	last_id at a point considerably later in the
64	0064	1	update operation.
65	0065	1	
66	0066	1	V03-025 JWT0183 Jim Teague 21-May-1984
67	0067	1	Fix cause of famous "RFA-accessed record deleted"
68	0068	1	error that occurred infrequently on \$UPDATES.
69	0069	1	It was simply a case of RMS not trying hard
70	0070	1	enough for the record. RMS first tries to find the
71	0071	1	record at the last place it saw it (LAST_ID and
72	0072	1	LAST_VBN) by calling FIND_BY_RFA. If someone
73	0073	1	has split the bucket out from under us, and the
74	0074	1	desired record described by the LAST fields was
75	0075	1	not in its original bucket, FIND_BY_RFA will tell
76	0076	1	us RMS\$DEL. If that is the case, RMS will have
77	0077	1	to simply follow the record from it's RRV in the
78	0078	1	original bucket using FIND_BY_RRV. It used to
79	0079	1	give up if FIND_BY_RFA failed with RMS\$DEL.
80	0080	1	
81	0081	1	V03-024 MCN0003 Maria del C. Nasr 04-Apr-1983
82	0082	1	Change linkage of RMSNULLKEY to RL\$JSB.
83	0083	1	
84	0084	1	V03-023 TMK0014 Todd M. Katz 26-Mar-1983
85	0085	1	Change the linkage for RMSRU_JOURNAL3 from RL\$RABREG_467 to
86	0086	1	RL\$RABREG_67.
87	0087	1	
88	0088	1	V03-022 MCN0002 Maria del C. Nasr 24-Mar-1983
89	0089	1	More linkages reorganization
90	0090	1	
91	0091	1	V03-021 RAS0135 Ron Schaefer 17-Mar-1983
92	0092	1	Fix spelling of RJR\$_UPDAT -> RJR\$_UPDATE.
93	0093	1	
94	0094	1	V03-020 TMK0013 Todd M. Katz 16-Mar-1983
95	0095	1	Change the linkage for RMSRU_JOURNAL3 from RL\$RABREG_67 to
96	0096	1	RL\$RABREG_467.
97	0097	1	
98	0098	1	V03-019 TMK0012 Todd M. Katz 16-Mar-1983
99	0099	1	Change the symbol RMSR_UPDAT to RJR_UPDAT.
100	0100	1	
101	0101	1	V03-018 MCN0001 Maria del C. Nasr 28-Feb-1983
102	0102	1	Reorganize linkages
103	0103	1	
104	0104	1	V03-017 TMK0011 Todd M. Katz 12-Jan-1983
105	0105	1	Add support for Recovery Unit Journalling and RU ROLLBACK
106	0106	1	Recovery of ISAM files. This involves modifications to
107	0107	1	RMS\$REPLACE, RMS\$UPDATE_SCAN, and RMS\$UPDATE3B such that:
108	0108	1	
109	0109	1	1. The RU Journalling is done before any permanent modifications
110	0110	1	are made to the file.
111	0111	1	
112	0112	1	2. No space is reclaimed as the result of \$UPDATES done within
113	0113	1	Recovery Units. In other words, if old SIDs are to be deleted
114	0114	1	they are just marked RU_DELETE, and no space is reclaimed.

```
115 0115 1 | Likewise, if the new primary data record is smaller than the
116 0116 1 | old record, the record is marked RU_UPDATE, it does not
117 0117 1 | shrink in size (in fact it may even grow by one byte), and
118 0118 1 | it is put into a special format so that the space maybe
119 0119 1 | reserved in case the Recovery Unit must be aborted.
120 0120 1 |
121 0121 1 | 3. If a Recovery Unit Rollback is in progress, there is no need
122 0122 1 | to insert new SIDs which are still represented in the file
123 0123 1 | (they are marked RU_DELETE). Instead the RU_DELETE bit is
124 0124 1 | cleared in the SID element's control byte.
125 0125 1 |
126 0126 1 | V03-016 TMK0010 Todd M. Katz 05-Jan-1983
127 0127 1 | RMSUPDATE3B was saving, zeroing, and then restoring the current
128 0128 1 | NRP key of reference while the old SIDs were being deleted.
129 0129 1 | This is no longer necessary.
130 0130 1 |
131 0131 1 | V03-015 TMK0009 Todd M. Katz 22-Dec-1982
132 0132 1 | I have changed a sufficient amount of $UPDATE, including the
133 0133 1 | basic algorithms, so that the audit trail has been invalidated.
134 0134 1 |
135 0135 1 | *****
136 0136 1 |
137 0137 1 | LIBRARY 'RMSLIB:RMS';
138 0138 1 |
139 0139 1 | REQUIRE 'RMSSRC:RMSIDXDEF';
140 0204 1 |
141 0205 1 | ! Define default psects for code
142 0206 1 |
143 0207 1 | PSECT
144 0208 1 |     CODE = RMSRMS3(PSECT_ATTR),
145 0209 1 |     PLIT = RMSRMS3(PSECT_ATTR);
146 0210 1 |
147 0211 1 | ! Linkages
148 0212 1 |
149 0213 1 | LINKAGE
150 0214 1 |     L_COMPARE_KEY,
151 0215 1 |     L_ERROR_LINK1,
152 0216 1 |     L_ERROR_LINK2,
153 0217 1 |     L_JSB,
154 0218 1 |     L_JSB01,
155 0219 1 |     L_LINK 7 10 11,
156 0220 1 |     L_PRESERVE1,
157 0221 1 |     L_RABREG,
158 0222 1 |     L_RABREG_4567,
159 0223 1 |     L_RABREG_567,
160 0224 1 |     L_RABREG_67,
161 0225 1 |     L_RABREG_7,
162 0226 1 |     L_REC_OVRD,
163 0227 1 |
164 0228 1 | ! Local Linkage
165 0229 1 |
166 0230 1 |     RL$REPLACE = JSB () :
167 0231 1 |         GLOBAL (R_BKT_ADDR, R_IRAB, R_IFAB, R_IDX_DFN, R_REC_ADDR),
168 0232 1 |     RL$UPDATE_SCAN = JSB () :
169 0233 1 |         GLOBAL (R_REC_ADDR, R_IMPURE, R_IDX_DFN, R_IFAB, R_IRAB, R_RAB);
170 0234 1 |
171 0235 1 | ! External Routines
```


: 172	0236	1	!		
: 173	0237	1	EXTERNAL ROUTINE		
: 174	0238	1	RMSCLEAN BDB	:	RL\$ERROR LINK1 NOVALUE,
: 175	0239	1	RMSCOMPARE KEY	:	RL\$COMPARE KEY,
: 176	0240	1	RMSDELETE_SIDR	:	RL\$RABREG_7,
: 177	0241	1	RMSEXPAND_KEYD	:	RL\$JSB01,
: 178	0242	1	RMSFIND_BY_RFA	:	RL\$RABREG_67,
: 179	0243	1	RMSFIND_BY_RRV	:	RL\$RABREG_67,
: 180	0244	1	RMSGET_NEXT KEY	:	RL\$LINK 7-10 11,
: 181	0245	1	RMSINS-ALL SIDR	:	RL\$RABREG_4567,
: 182	0246	1	RMSINSERT_ODR	:	RL\$RABREG_4567,
: 183	0247	1	RMSINSS OR IDX	:	RL\$RABREG_567,
: 184	0248	1	RMSKEY DESC	:	RL\$RABREG_7,
: 185	0249	1	RMSMOVE	:	RL\$PRESERVE1,
: 186	0250	1	RMSNULLKEY	:	RL\$JSB,
: 187	0251	1	RMSPACK_REC	:	RL\$RABREG_567,
: 188	0252	1	RMSPUT_OPD_CHKS	:	RL\$RABREG_7,
: 189	0253	1	RMSPUT_OPD_ERROR	:	RL\$ERROR LINK2 NOVALUE,
: 190	0254	1	RMSPUT_OPD_FIN	:	RL\$RABREG,
: 191	0255	1	RMSPUT_OPD_SPL	:	RL\$RABREG_4567,
: 192	0256	1	RMSRECORD_ID	:	RL\$RABREG_67,
: 193	0257	1	RMSRECORD_KEY	:	RL\$PRESERVE1,
: 194	0258	1	RMSRECORD_VBN	:	RL\$PRESERVE1,
: 195	0259	1	RMSREC_OVHD	:	RL\$REC_OVHD,
: 196	0260	1	RMSRLSBKT	:	RL\$PRESERVE1,
: 197	0261	1	RMSRU JOURNAL3	:	RL\$RABREG_67,
: 198	0262	1	RMSUNPACK_REC	:	RL\$JSB01,
: 199	0263	1	RMSUPDEL COM	:	RL\$RABREG_67;

RMSREPLACE

```
201 0264 1 XSBTTL 'RMSREPLACE'
202 0265 1 ROUTINE RMSREPLACE (NEW_REC_SIZE) : RL$REPLACE =
203 0266 1
204 0267 1 ++
205 0268 1
206 0269 1 FUNCTIONAL DESCRIPTION:
207 0270 1
208 0271 1 This routine updates the record pointed to by REC_ADDR if the new record is
209 0272 1 the same size or smaller than record being updated. Otherwise, this routine
210 0273 1 deletes entirely the old record thus setting up the conditions for performing
211 0274 1 a bucket split.
212 0275 1
213 0276 1 If the $UPDATE is taking place within a Recovery Unit and the new record is
214 0277 1 smaller than the old record by more than one byte then the record is marked
215 0278 1 RU_UPDATE and placed into a special format so that the old size maybe reserved
216 0279 1 in case it become necessary to ROLLBACK recover the Recovery Unit. This
217 0280 1 special format is to have the size of the amount of space that is being
218 0281 1 reserved in the primary data record overhead's record size field while the
219 0282 1 true size of the record occupies the last two bytes of this reserved space.
220 0283 1
221 0284 1 If the $UPDATE is taking place within a Recovery Unit and the new record is
222 0285 1 smaller than the old record by exactly one byte, then the primary data record
223 0286 1 must also be marked RU_UPDATE and placed into the special format. However, as
224 0287 1 placing the record in a special format would require it to grow in size by one
225 0288 1 byte, this routine does not replace the old with the new record. Instead, the
226 0289 1 old record is deleted, the state bit IRB$V_RU_UPDATE is set, and a 0 is
227 0290 1 returned indicating that a strict replacement could not be done. When the new
228 0291 1 record is eventually inserted, the state bit IRB$V_RU_UPDATE indicates that
229 0292 1 its size must be increased by two, and that it should be put into this special
230 0293 1 format.
231 0294 1
232 0295 1 If the $UPDATE is taking place as a RU ROLLBACK Recovery operation and the
233 0296 1 new record is smaller than the old record by more than one byte, then the
234 0297 1 primary data record is also marked RU_UPDATE and is placed into the special
235 0298 1 format. This special formatting is not necessary if the new record is exactly
236 0299 1 one byte smaller than the old record.
237 0300 1
238 0301 1 CALLING SEQUENCE:
239 0302 1
240 0303 1 RMSREPLACE()
241 0304 1
242 0305 1 INPUT PARAMETERS:
243 0306 1
244 0307 1 NEW_REC_SIZE - size of the new record
245 0308 1
246 0309 1 IMPLICIT INPUTS:
247 0310 1
248 0311 1 BKT_ADDR - address of the primary data bucket
249 0312 1 BKT$W_FREESPACE - freespace offset pointer
250 0313 1
251 0314 1 IDX_DFN - address of the primary key index descriptor
252 0315 1 IDX$V_KEY_COMPR - if set, primary key compression is enabled
253 0316 1
254 0317 1 IFAB - address of the IFAB
255 0318 1 IFB$W_KBUFSZ - size of a keybuffer
256 0319 1 IFB$B_PLG_VER - prologue version of the file
257 0320 1 IFB$V_RUP - if set, Recovery Unit is in progress
```



```
258 0321 1 IFBSV_RU_RECVR - if set, RU ROLLBACK Recovery in progress
259 0322 1
260 0323 1 IRAB - address of the IRAB
261 0324 1 IRBSL_KEYBUF - address of the contiguous keybuffers
262 0325 1 IRBSL_RECBUF - address of the record buffer
263 0326 1 IRBSL_RBF - address of the user's record buffer
264 0327 1
265 0328 1 REC_ADDR - address of the primary data record
266 0329 1
267 0330 1 OUTPUT PARAMETERS:
268 0331 1 NONE
269 0332 1
270 0333 1 IMPLICIT OUTPUTS:
271 0334 1
272 0335 1 BKT_ADDR - address of the primary data bucket
273 0336 1 BKT$W_FREESPACE - freespace offset pointer
274 0337 1
275 0338 1 IRAB - address of IRAB
276 0339 1 IRBSV_RU_UPDATE - if set, build new record in special format
277 0340 1
278 0341 1 ROUTINE VALUE:
279 0342 1
280 0343 1 0 - if the new record did not replace the old record
281 0344 1 1 - if the new record replaced the old record
282 0345 1
283 0346 1 SIDE EFFECTS:
284 0347 1
285 0348 1 AP and keybuffer 5 are trashed.
286 0349 1
287 0350 1 If STATUS = 0, then the old primary data record has been deleted, the
288 0351 1 bucket's freespace offset pointer has been updated to reflect this,
289 0352 1 and keybuffer 3 contains the primary key of the record.
290 0353 1 If STATUS = 1, then the old primary data record has been replaced by the
291 0354 1 new record, and any difference in size between the two is reflected
292 0355 1 both within the record's size field, and in the bucket's freespace
293 0356 1 offset pointer.
294 0357 1
295 0358 1 --
296 0359 1
297 0360 2 BEGIN
298 0361 2
299 0362 2 EXTERNAL REGISTER
300 0363 2 R_BKT_ADDR_STR,
301 0364 2 R_IDX_DFN_STR,
302 0365 2 R_IFAB_STR,
303 0366 2 R_IRAB_STR,
304 0367 2 R_REC_ADDR_STR;
305 0368 2
306 0369 2 GLOBAL REGISTER
307 0370 2 R_RAB,
308 0371 2 R_IMPURE,
309 0372 2 R_BDB;
310 0373 2
311 0374 2 BUILTIN
312 0375 2 AP;
313 0376 2
314 0377 2 LOCAL
```



```
315 0378 2 DIFFERENCE : SIGNED,  
316 0379 RECORD_OVHD,  
317 0380 STATUS,  
318 0381 OLD_REC_SIZE,  
319 0382 OLD_REC_ADDR;  
320 0383  
321 0384 ! Retrieve the size of the old record.  
322 0385  
323 0386 BEGIN  
324 0387  
325 0388 LOCAL  
326 0389 REC_SIZE;  
327 0390  
328 0391 RECORD_OVHD = RMSREC_OVHD(0; REC_SIZE);  
329 0392 OLD_REC_SIZE = .REC_SIZE;  
330 0393 OLD_REC_ADDR = .REC_ADDR + .RECORD_OVHD;  
331 0394 END;  
332 0395  
333 0396 ! If the key is compressed, save in key buffer 5 the key of the record  
334 0397 ! that will be deleted, including the compression overhead. This might  
335 0398 ! be used by RM$EXPAND_KEYD later.  
336 0399  
337 0400 IF .IDX_DFN[IDX$V_KEY_COMPR]  
338 0401 THEN  
339 0402 RMSMOVE ((.OLD_REC_ADDR)<0,8> + 2, .OLD_REC_ADDR, KEYBUF_ADDR(5));  
340 0403  
341 0404 ! Figure out what the difference in records sizes is.  
342 0405  
343 0406 DIFFERENCE = .OLD_REC_SIZE - .NEW_REC_SIZE;  
344 0407  
345 0408 ! If the record will grow in size as a result of being updated, setup  
346 0409 ! so the old record can be removed entirely from the bucket and for  
347 0410 ! the split of the primary data bucket in case it becomes necessary.  
348 0411  
349 0412 ! If the record will shrink in size by one byte as a result of being updated  
350 0413 ! and the process is currently in a Recovery Unit, then also setup so that  
351 0414 ! the old record can be removed entirely from the bucket. In addition, set  
352 0415 ! the state bit IRB$V_RU_UPDATE so that when this record gets built, it  
353 0416 ! will be built in the special format that is required.  
354 0417  
355 0418 IF .DIFFERENCE LSS 0  
356 0419 OR  
357 0420 (.IFAB[IFB$V_RUP]  
358 0421 AND  
359 0422 .DIFFERENCE EQL 1)  
360 0423 THEN  
361 0424 BEGIN  
362 0425 IF .IFAB[IFB$V_RUP]  
363 0426 AND  
364 0427 .DIFFERENCE EQL 1  
365 0428 THEN  
366 0429 IRAB[IRB$V_RU_UPDATE] = 1;  
367 0430  
368 0431  
369 0432 DIFFERENCE = .OLD_REC_ADDR + .OLD_REC_SIZE - .REC_ADDR;  
370 0433 OLD_REC_ADDR = .REC_ADDR;  
371 0434
```

```
372 0435 ! Get primary key value into keybuffer 3 for use by RM$PUT_UPD_SPL.
373 0436
374 0437 REC_ADDR = .IRAB[IRBSL_RBF];
375 0438
376 0439 AP = 3;
377 0440 RMSRECORD_KEY(KEYBUF_ADDR(3));
378 0441
379 0442 STATUS = 0;
380 0443 REC_ADDR = .OLD_REC_ADDR;
381 0444 END
382 0445
383 0446 ! The new record will be able to fit into the space currently occupied
384 0447 by the old record either because the records are the same size, or the
385 0448 new record is smaller than the old record.
386 0449
387 0450 ELSE
388 0451 BEGIN
389 0452 LOCAL
390 0453 BUFFER;
391 0454
392 0455 ! Assume that the record will not be in special format and clear the
393 0456 appropriate record control bit.
394 0457
395 0458 REC_ADDR[IRCSV_RU_UPDATE] = 0;
396 0459
397 0460 ! The new record is smaller than the old record, so fix the record size.
398 0461
399 0462 IF .DIFFERENCE NEQ 0
400 0463 THEN
401 0464
402 0465 ! If the process is not currently within a Recovery Unit or the
403 0466 operation is not a RU ROLLBACK Recovery operation; or, if the
404 0467 current operation is a RU ROLLBACK Recovery operation but the
405 0468 record's size would only decrease by one byte, then just move
406 0469 in the new size into the record overhead's size field.
407 0470
408 0471 IF NOT (.IFAB[IFBSV_RUP]
409 0472 OR
410 0473 .IFAB[IFBSV_RU_RECVR])
411 0474 OR
412 0475 (.IFAB[IFBSV_RU_RECVR]
413 0476 AND
414 0477 .DIFFERENCE EQLU 1)
415 0478 THEN
416 0479 (.OLD_REC_ADDR - IRCSC_DATSZFLD)<0, 16> = .NEW_REC_SIZE
417 0480
418 0481 ! If the process is currently within a Recovery Unit then leave the
419 0482 record size field in the record overhead alone, and instead move
420 0483 the new size into the last two bytes in the record's reserved
421 0484 space. Set the RU_UPDATE bit in the record's control byte to
422 0485 indicate that the record is in a special format, and zero
423 0486 DIFFERENCE so that no space is reclaimed from the record.
424 0487
425 0488 This sequence is also done if the current operation is a RU
426 0489 ROLLBACK Recovery operation and the record shrinks by more than
427 0490 one byte.
428 0491
```

```
      ELSE
      BEGIN
        REC_ADDR[IRCSV_RU_UPDATE] = 1;
        DIFFERENCE = 0;
        (.OLD_REC_ADDR + .OLD_REC_SIZE
         - IRCSC_DATSZFLD)<0,16> = .NEW_REC_SIZE;
      END;

      ! Move in the new record. Where the new record is moved in from is
      ! prologue dependent.
      IF .IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3
      THEN
        BUFFER = .IRAB[IRB$L_RBF]
      ELSE
        BUFFER = .IRAB[IRB$L_RECBUF];

      OLD_REC_ADDR = RMSMOVE (.NEW_REC_SIZE, .BUFFER, .OLD_REC_ADDR);

      STATUS = 1;
      END;

      ! If the new record grew in size delete it from the bucket and fix up the
      ! bucket's freespace offset pointer. If the record shrunk in size, then
      ! shuffle all the trailing records in the bucket to take up this available
      ! space and fix up the bucket's freespace offset pointer.
      BEGIN
      LOCAL
        LENGTH;

      IF .DIFFERENCE NEQ 0
      THEN
      BEGIN
        BKT_ADDR[BKT$W_FREESPACE] = .BKT_ADDR[BKT$W_FREESPACE]
          - .DIFFERENCE<0, 16>;

        LENGTH = .BKT_ADDR[BKT$W_FREESPACE] - (.OLD_REC_ADDR
          - .BKT_ADDR)<0, 16>;

        IF .LENGTH GTR 0
        THEN
          RMSMOVE (.LENGTH, (.OLD_REC_ADDR + .DIFFERENCE), .OLD_REC_ADDR);

        END;

        ! If STATUS is 0, then the record grew in size, it has been deleted from
        ! the bucket, and a regular record insert must be done. If the key is
        ! compressed, the key of the following record (if there is one) must be
        ! expanded to reflect the deleted record.
        IF .IDX_DFN[IDX$V_KEY_COMPR]
        AND
        .STATUS EQLU 0
```



```

486      0549      AND
487      0550      .LENGTH GTR 0
488      0551      AND
489      0552      NOT .REC_ADDR[IRCSV_RRV]
490      0553      THEN
491      0554      RM$EXPAND_KEYD (KEYBUF_ADDR(5), .REC_ADDR + .RECORD_OVHD);
492      0555
493      0556      END;
494      0557
495      0558      RETURN .STATUS;
496      0559
497      0560      END;
```

```
.TITLE RM3UPDATE
.IDENT \V04-000\
```

```
.EXTRN RM$CLEAN_BDB, RM$COMPARE_KEY
.EXTRN RM$DELETE_SIDR, RM$EXPAND_KEYD
.EXTRN RM$FIND_BY_RFA, RM$FIND_BY_RRV
.EXTRN RM$GET_NEXT_KEY
.EXTRN RM$INS_ALL_SIDR
.EXTRN RM$INSERT_ODR, RM$INSS_OR_IDX
.EXTRN RM$KEY_DESC, RM$MOVE
.EXTRN RM$NUL[KEY, RM$PACK_REC
.EXTRN RM$PUT_UPD_CHKS
.EXTRN RM$PUT_UPD_ERROR
.EXTRN RM$PUT_UPD_FIN, RM$PUT_UPD_SPL
.EXTRN RM$RECORD_ID, RM$RECORD_KEY
.EXTRN RM$RECORD_VBN, RM$REC_OVHD
.EXTRN RM$RLSBKT, RM$RU_JOURNAL3
.EXTRN RM$UNPACK_REC, RM$UPDDELCOM
```

```
.PSECT RMSRMS3,NOWRT, GBL, PIC,2
```

			091C	BF	BB	00000	RMSREPLACE:		
		5E		08	C2	00004	PUSHR	#M<R2,R3,R4,R8,R11>	0265
				51	D4	00007	SUBL2	#8, SP	0391
				0000G	30	00009	CLRL	R1	
		6E		50	D0	0000C	BSBW	RM\$REC_OVHD	
		53		51	D0	0000F	MOVL	R0, RECORD_OVHD	0392
51		56		6E	C1	00012	MOVL	REC_SIZE, OLD_REC_SIZE	0393
17	1C	A7		06	E1	00016	ADDL3	RECORD_OVHD, REC_ADDR, OLD_REC_ADDR	0400
		50	00B4	CA	3C	0001B	BBC	#6, 28(IDX_DFN), 1\$	0402
			60	B940	DF	00020	MOVZWL	180(IFAB), R0	
				51	DD	00024	PUSHAL	@96(IRAB)[R0]	
		7E		61	9A	00026	PUSHL	OLD_REC_ADDR	
		6E		02	C0	00029	MOVZBL	(OLD_REC_ADDR), -(SP)	
				0000G	30	0002C	ADDL2	#2, TSP	
		5E		0C	C0	0002F	BSBW	RM\$MOVE	
52		53	20	AE	C3	00032	ADDL2	#12, SP	0406
				0B	19	00037	SUBL3	NEW_REC_SIZE, OLD_REC_SIZE, DIFFERENCE	0418
3E	00A2	CA		02	E1	00039	BLSS	2\$	0420
		01		52	D1	0003F	BBC	#2, 162(IFAB), 4\$	0422
				39	12	00042	CMPL	DIFFERENCE, #1	
0A	00A2	CA		02	E1	00044	BNEQ	4\$	
							BBC	#2, 162(IFAB), 3\$	0426

		01		52	D1	0004A	CMPL	DIFFERENCE, #1	0428
				05	12	0004D	BNEQ	3\$	
	07	A9	80	8F	88	0004F	BISB2	#128, 7(IRAB)	0430
50		51		53	C1	00054	ADDL3	OLD_REC_SIZE, OLD_REC_ADDR, R0	0432
52		50		56	C3	00058	SUBL3	REC_ADDR, R0, DIFFERENCE	
		51		56	D0	0005C	MOVL	REC_ADDR, OLD_REC_ADDR	0433
		56	58	A9	D0	0005F	MOVL	88(IRAB), REC_ADDR	0437
		5C		03	D0	00063	MOVL	#3, AP	0439
		50	00B4	CA	3C	00066	MOVZWL	180(IFAB), R0	0440
			60	B940	3F	0006B	PUSHAW	296(IRAB)[R0]	
				0000G	30	0006F	BSBW	RMSRECORD_KEY	
		5E		04	C0	00072	ADDL2	#4, SP	
			04	AE	D4	00075	CLRL	STATUS	0442
		56		51	D0	00078	MOVL	OLD_REC_ADDR, REC_ADDR	0443
				55	11	00078	BRB	11\$	0418
		66	40	8F	8A	0007D	BICB2	#64, (REC_ADDR)	0459
				52	D5	00081	TSTL	DIFFERENCE	0463
				2A	13	00083	BEQL	8\$	
05	00A2	CA		02	E0	00085	BBS	#2, 162(IFAB), 5\$	0472
		0A	00A1	CA	E9	00088	BLBC	161(IFAB), 6\$	0474
		0C	00A1	CA	E9	00090	BLBC	161(IFAB), 7\$	0476
		01		52	D1	00095	CMPL	DIFFERENCE, #1	0478
				07	12	00098	BNEQ	7\$	
	FE	A1	20	AE	B0	0009A	MOVW	NEW_REC_SIZE, -2(OLD_REC_ADDR)	0480
				0E	11	0009F	BRB	8\$	
		66	40	8F	88	000A1	BISB2	#64, (REC_ADDR)	0495
				52	D4	000A5	CLRL	DIFFERENCE	0496
			FE	A341	9F	000A7	PUSHAB	-2(OLD_REC_SIZE)[OLD_REC_ADDR]	0498
		9E	24	AE	B0	000AB	MOVW	NEW_REC_SIZE, 2(SP)+	
		03	00B7	CA	91	000AF	CMPL	183(IFAB), #3	0504
				06	1E	000B4	BGEQU	9\$	
		50	58	A9	D0	000B6	MOVL	88(IRAB), BUFFER	0506
				04	11	000BA	BRB	10\$	
		50	68	A9	D0	000BC	MOVL	104(IRAB), BUFFER	0508
				03	BB	000C0	PUSHR	#*M<R0,R1>	0510
			28	AE	DD	000C2	PUSHL	NEW_REC_SIZE	
				0000G	30	000C5	BSBW	RMSMOVE	
		5E		0C	C0	000C8	ADDL2	#12, SP	
		51		50	D0	000CB	MOVL	R0, OLD_REC_ADDR	
	04	AE		01	D0	000CE	MOVL	#1, STATUS	0512
				52	D5	000D2	TSTL	DIFFERENCE	0525
				1E	13	000D4	BEQL	12\$	
	04	A5		52	A2	000D6	SUBW2	DIFFERENCE, 4(BKT_ADDR)	0530
50		55		51	C3	000DA	SUBL3	OLD_REC_ADDR, BKT_ADDR, R0	0533
		53	04	A5	3C	000DE	MOVZWL	4(BKT_ADDR), LENGTH	
		53		50	C0	000E2	ADDL2	R0, LENGTH	
				0D	15	000E5	BLEQ	12\$	0535
				51	DD	000E7	PUSHL	OLD_REC_ADDR	0537
			6241	9F	000E9		PUSHAB	(DIFFERENCE)[OLD_REC_ADDR]	
			53	DD	000EC		PUSHL	LENGTH	
			0000G	30	000EE		BSBW	RMSMOVE	
		5E		0C	C0	000F1	ADDL2	#12, SP	
1E	1C	A7		06	E1	000F4	BBC	#6, 28(IDX_DFN), 13\$	0546
			04	AE	D5	000F9	TSTL	STATUS	0548
				19	12	000FC	BNEQ	13\$	
				53	D5	000FE	TSTL	LENGTH	0550
				15	15	00100	BLEQ	13\$	


```
500 0562 1 XSBTTL 'RMSUPDATE_SCAN'
501 0563 1 ROUTINE RMSUPDATE_SCAN : RLSUPDATE_SCAN =
502 0564 1
503 0565 1 ++
504 0566 1
505 0567 1 FUNCTIONAL DESCRIPTION:
506 0568 1
507 0569 1 Compare the old and new records to determine which new keys are to be
508 0570 1 inserted and which old keys are to be deleted. If the file is a prologue
509 0571 1 3 file, then unpack the primary data record into the old record buffer,
510 0572 1 and if there are old alternate keys to be deleted and the file is a
511 0573 1 prologue 1 or 2 file, move the primary data record into the old record
512 0574 1 buffer as well.
513 0575 1
514 0576 1 CALLING SEQUENCE:
515 0577 1
516 0578 1 RMSUPDATE_SCAN()
517 0579 1
518 0580 1 INPUT PARAMETERS:
519 0581 1 NONE
520 0582 1
521 0583 1 IMPLICIT INPUTS:
522 0584 1
523 0585 1 IDX_DFN - address of the primary key index descriptor
524 0586 1 IDXSV_CHGKEYS - if set, keys are allowed to change
525 0587 1 IDXSDESC_NO - descriptor number
526 0588 1 IDXSKEYSZ - size of key
527 0589 1 IDXS_MINRECSZ - minimum size of record containing this key
528 0590 1
529 0591 1 IFAB - address of IFAB
530 0592 1 IFBSB_PLG_VER - prologue version of file
531 0593 1
532 0594 1 IRAB - address of IRAB
533 0595 1 IRBSB_CUR_KREF - current NRP key of reference
534 0596 1 IRBSL_OLDBUF - address of old record buffer
535 0597 1 IRBSL_RBF - address of new record buffer
536 0598 1 IRBSW_RSZ - size of new record
537 0599 1 IRBSV_UPDATE_IF - if set, $PUT converted into an $UPDATE
538 0600 1 IRBSL_UPDBUF - address of update buffer
539 0601 1
540 0602 1 REC_ADDR - address of primary data record
541 0603 1
542 0604 1 OUTPUT PARAMETERS:
543 0605 1 NONE
544 0606 1
545 0607 1 IMPLICIT OUTPUTS:
546 0608 1
547 0609 1 IRAB - address of IRAB
548 0610 1 IRBSV_FIND_LAST - set to 0
549 0611 1
550 0612 1 ROUTINE VALUE:
551 0613 1
552 0614 1 CHG - if primary key changes key value at all, or any alternate key
553 0615 1 changes values and CHGKEYS not set
554 0616 1 SUC - otherwise
555 0617 1
556 0618 1 SIDE EFFECTS:
```

```
557 0619 1 |
558 0620 1 | AP is trashed.
559 0621 1 | Each byte in the update buffer corresponds to an index with:
560 0622 1 |     INS_NEW set if a new key value is to be inserted
561 0623 1 |     OLD_DEL set if the old key value requires deletion.
562 0624 1 | The same flags for the primary key byte indicate whether any
563 0625 1 |     alternate keys require insertion or deletion at all.
564 0626 1 | If the file is a prologue 3 file then the old record has been unpacked
565 0627 1 |     into the old record buffer.
566 0628 1 | If the file is a prologue 1 or 2 file, and there are old alternate keys
567 0629 1 |     to be deleted then the old record has been moved into the old record
568 0630 1 |     buffer.
569 0631 1 |
570 0632 1 | --
571 0633 1 |
572 0634 1 | BEGIN
573 0635 1 |
574 0636 1 | EXTERNAL REGISTER
575 0637 1 |     R_IDX_DFN_STR,
576 0638 1 |     R_IFAB_STR,
577 0639 1 |     R_IMPURE,
578 0640 1 |     R_IRAB_STR,
579 0641 1 |     R_RAB,
580 0642 1 |     R_REC_ADDR_STR;
581 0643 1 |
582 0644 1 | BUILTIN
583 0645 1 |     AP;
584 0646 1 |
585 0647 1 | LOCAL
586 0648 1 |     OLD_REC_ADDR,
587 0649 1 |     OLD_REC_SIZE : WORD;
588 0650 1 |
589 0651 1 | ! Retrieve the size of the old record, as it is in the bucket, and set
590 0652 1 | ! up a pointer to the old data record itself.
591 0653 1 |
592 0654 1 | BEGIN
593 0655 1 |
594 0656 1 | LOCAL
595 0657 1 |     REC_SIZE;
596 0658 1 |
597 0659 1 | OLD_REC_ADDR = .REC_ADDR + RMSREC_OVHD(0; REC_SIZE);
598 0660 1 | OLD_REC_SIZE = .REC_SIZE;
599 0661 1 | END;
600 0662 1 |
601 0663 1 | ! If the file is a prologue 3 file, then the old record must be unpacked
602 0664 1 | ! into the old record buffer. Reset the pointer to the data portion of the
603 0665 1 | ! old record so that it points into the old record buffer.
604 0666 1 |
605 0667 1 | IF .IFAB[IFB$B_PLG_VER] EQLU PLG$C_VER_3
606 0668 1 | THEN
607 0669 1 |     BEGIN
608 0670 1 |
609 0671 1 |     GLOBAL REGISTER
610 0672 1 |         R_BKT_ADDR;
611 0673 1 |
612 0674 1 |     LOCAL
613 0675 1 |         SAVE_REC_ADDR : REF BBLOCK;
```

```
0676
0677
0678
0679
0680
0681
0682
0683
0684
0685
0686
0687
0688
0689
0690
0691
0692
0693
0694
0695
0696
0697
0698
0699
0700
0701
0702
0703
0704
0705
0706
0707
0708
0709
0710
0711
0712
0713
0714
0715
0716
0717
0718
0719
0720
0721
0722
0723
0724
0725
0726
0727
0728
0729
0730
0731
0732

SAVE REC_ADDR = .REC_ADDR;
REC_ADDR = .OLD_REC_ADDR;
OLD_REC_ADDR = .IRAB[IRBSL_OLDBUF] + 2;

! If this record is marked RU_UPDATE then the last two bytes of the
! reserved space of the record contain the actual size of the record.
IF .SAVE_REC_ADDR[IRCSV_RU_UPDATE]
THEN
    OLD_REC_SIZE = .(.REC_ADDR + .OLD_REC_SIZE - IRCSC_DATSZFLD)<0,16>;

! If this operation is a $PUT converted into an $UPDATE, then the
! primary key of the old record maybe found in keybuffer 2. Likewise,
! if this is a $UPDATE and RMS positioned to the record by means of the
! primary key of reference although not by a $FIND, then the primary
! key of the old record maybe found in keybuffer 1. Otherwise, a bucket
! scan must be done to re-expand the primary key.
IF .IRAB[IRBSV_UPDATE_IF]
THEN
    AP = 2
ELSE
    IF .IRAB[IRBSB_CUR_KREF] EQLU 0
        AND
        NOT .IRAB[IRBSV_FIND_LAST]
    THEN
        AP = 1
    ELSE
        AP = 0;

OLD_REC_SIZE = RMSUNPACK_REC (.OLD_REC_ADDR, .OLD_REC_SIZE);
(.IRAB[IRBSL_OLDBUF])<0,16> = .OLD_REC_SIZE;
REC_ADDR = .SAVE_REC_ADDR
END;

IRAB[IRBSV_FIND_LAST] = 0;

! If the primary key has changed, return an error. This comparison does
! not have to be done if the operation is an UPDATE_IF.
IF NOT .IRAB[IRBSV_UPDATE_IF]
THEN
    BEGIN
        AP = 0;

        IF RMSCOMPARE_KEY (.OLD_REC_ADDR,
                           .IRAB[IRBSL_RBF],
                           .IDX_DFN[IDX$B_KEYSZ])
        THEN
            RETURN RMSERR (CHG);
        END;

! The INS_NEW and OLD_DEL flags for the first byte of the update buffer
! (which would be for key 0) are used to indicate whether any inserts or
! deletes are to be done at all. Zero these.
```



```
671 0733 2 BBLOCK[.IRAB[IRBSL_UPDBUF], UPD$B_FLAGS] = 0;
672 0734 2
673 0735 2 ! Process all of the alternate keys determining whether the alternate key has
674 0736 2 ! changed, the old key must be deleted, and a new key must be inserted.
675 0737 2
676 0738 2 WHILE RMSGET_NEXT_KEY()
677 0739 2 DO
678 0740 2 BEGIN
679 0741 2
680 0742 2 LOCAL
681 0743 2 UPD_BUF_ADDR : REF BBLOCK;
682 0744 2
683 0745 2 ! Point to the update buffer associated with this index.
684 0746 2
685 0747 2 UPD_BUF_ADDR = .IRAB[IRBSL_UPDBUF] + .IDX_DFN[IDX$B_DESC_NO];
686 0748 2 UPD_BUF_ADDR[UPD$B_FLAGS] = 0;
687 0749 2
688 0750 2 AP = 0;
689 0751 2
690 0752 2 ! Set the flag INS_NEW for this key of reference if the key is present
691 0753 2 ! in the new record, disregarding for the time being whether the key
692 0754 2 ! is present in the old record and whether the two keys are the same.
693 0755 2
694 0756 2 IF .IRAB[IRBSW_RSZ] GEQU .IDX_DFN[IDX$W_MINRECSZ]
695 0757 2 AND
696 0758 2 RMSNULLKEY (.IRAB[IRBSL_RBF])
697 0759 2 THEN
698 0760 2 UPD_BUF_ADDR[UPD$V_INS_NEW] = 1;
699 0761 2
700 0762 2 ! Determine whether this same key was present in the old record.
701 0763 2
702 0764 2 IF .OLD_REC_SIZE GEQU .IDX_DFN[IDX$W_MINRECSZ]
703 0765 2 AND
704 0766 2 RMSNULLKEY (.OLD_REC_ADDR)
705 0767 2 THEN
706 0768 2
707 0769 2 ! If the key was present in the old record but is not present in
708 0770 2 ! the new record then flag that the old key for this key of
709 0771 2 ! reference should be deleted.
710 0772 2
711 0773 2 IF NOT .UPD_BUF_ADDR[UPD$V_INS_NEW]
712 0774 2 THEN
713 0775 2 UPD_BUF_ADDR[UPD$V_OLD_DEL] = 1
714 0776 2
715 0777 2 ! If the key is present in both records then compare the two keys.
716 0778 2
717 0779 2 ELSE
718 0780 2 BEGIN
719 0781 2 AP = 0;
720 0782 2
721 0783 2 ! If there are keys present in both records but they are
722 0784 2 ! different then flag that there is a new key to be inserted
723 0785 2 ! (already has been done) and an old key to be deleted for
724 0786 2 ! this key of reference.
725 0787 2
726 0788 2 IF RMSCOMPARE_KEY (.OLD_REC_ADDR,
727 0789 2 .IRAB[IRBSL_RBF],
```

```

728      0790 4
729      0791 4
730      0792 4
731      0793 4
732      0794 4
733      0795 4
734      0796 4
735      0797 4
736      0798 4
737      0799 4
738      0800
739      0801
740      0802
741      0803
742      0804
743      0805
744      0806
745      0807
746      0808
747      0809
748      0810
749      0811
750      0812
751      0813
752      0814
753      0815
754      0816
755      0817
756      0818
757      0819
758      0820
759      0821
760      0822
761      0823
762      0824
763      0825
764      0826
765      0827
766      0828
767      0829
768      0830
769      0831
770      0832
771      0833
772      0834
773      0835
774      0836
775      0837
776      0838
777      0839
778      0840 1

      .IDX_DFN[IDX$B_KEYSZ])
      THEN
        UPD_BUF_ADDR[UPD$V_OLD_DEL] = 1
        ! If there are keys present in both records and they are
        ! identical, then there is no need to insert or delete any
        ! keys for this key of reference.
      ELSE
        UPD_BUF_ADDR[UPD$V_INS_NEW] = 0;
      END;

      ! If the keys have changed between the two records, and this key of
      ! reference does not allow keys to change, return an error.
      IF .UPD_BUF_ADDR[UPD$B_FLAGS] NEQU 0
        AND
        NOT .IDX_DFN[IDX$V_CHGKEYS]
      THEN
        RETURN RMSERR (CHG);

      ! Summarize the key 0 flags to this point in the key-by-key comparison.
      BBLOCK[.IRAB[IRB$L_UPDBUF], UPD$B_FLAGS] =
        .BBLOCK[.IRAB[IRB$L_UPDBUF], UPD$B_FLAGS]
        OR
        .UPD_BUF_ADDR[UPD$B_FLAGS]

      END;

      ! If there are any old keys which will have to be deleted, and this is a
      ! prologue 1 or 2 file, then save the old record in the old record buffer.
      IF .BBLOCK[.IRAB[IRB$L_UPDBUF], UPD$V_OLD_DEL]
        AND
        .IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3
      THEN
        BEGIN
          GLOBAL REGISTER
            R_BDB;

          RMSMOVE (.OLD_REC_SIZE, .OLD_REC_ADDR, .IRAB[IRB$L_OLDDBUF] + 2);
          (.IRAB[IRB$L_OLDDBUF])<0,16> = .OLD_REC_SIZE;
        END;

      ! Reset to primary key descriptor and return success.
      RETURN RMSKEY_DESC(0)
    END;
```

3C BB 00000 RMSUPDATE_SCAN:

			51	D4	00002	PUSHR	#*M<R2,R3,R4,R5>	0563
			0000G	30	00004	CLRL	R1	0659
54	50		56	C1	00007	BSBW	RMSREC OVHD	
	52		51	B0	0000B	ADDL3	REC_ADDR, R0, OLD_REC_ADDR	
	03	00B7	CA	91	0000E	MOVW	REC_SIZE, OLD_REC_SIZE	0660
			48	12	00013	CMPB	1837(IRAB), #3	0667
	53		56	D0	00015	BNEQ	5\$	
	56		54	D0	00018	MOVL	REC_ADDR, SAVE_REC_ADDR	0677
54	6C		02	C1	0001B	MOVL	OLD_REC_ADDR, REC_ADDR	0678
0A	63		06	E1	00020	ADDL3	#2, 1087(IRAB), OLD_REC_ADDR	0679
	50		52	3C	00024	BBC	#6, (SAVE_REC_ADDR), 1\$	0684
		FE	A046	9F	00027	MOVZWL	OLD_REC_SIZE, R0	0686
	52		9E	B0	0002B	PUSHAB	-2(R0)[REC_ADDR]	
05	06		04	E1	0002E	MOVW	@(SP)+, OLD_REC_SIZE	
	5C		02	D0	00033	BBC	#4, 6(IRAB), 2\$	0695
			12	11	00036	MOVL	#2, AP	0697
		00C3	C9	95	00038	BRB	4\$	
			0A	12	0003C	TSTB	195(IRAB)	0699
05	04		05	E0	0003E	BNEQ	3\$	
	5C		01	D0	00043	BBS	#5, 4(IRAB), 3\$	0701
			02	11	00046	MOVL	#1, AP	0703
			5C	D4	00048	BRB	4\$	
	51		52	3C	0004A	CLRL	AP	0705
	50		54	D0	0004D	MOVZWL	OLD_REC_SIZE, R1	0707
		0000G	50	B0	00050	MOVL	OLD_REC_ADDR, R0	
			52	B0	00053	BSBW	RMSONPACK_REC	
	6C		53	D0	0005A	MOVW	R0, OLD_REC_SIZE	
			20	8A	0005D	MOVW	OLD_REC_SIZE, @108(IRAB)	0708
13	04		04	E0	00061	MOVL	SAVE_REC_ADDR, REC_ADDR	0709
	06		5C	D4	00066	BICB2	#32, 4(IRAB)	0712
			A7	9A	00068	BBS	#4, 6(IRAB), 6\$	0717
	53	20	A9	D0	0006C	CLRL	AP	0720
	51	58	54	D0	00070	MOVZBL	32(IDX_DFN), R0	0722
		0000G	50	E8	00076	MOVL	88(IRAB), R3	
	63		B9	94	00079	MOVL	OLD_REC_ADDR, R1	
		0000G	30	0007C	BSBW	RMSCOMPARE_KEY		
			50	E9	0007F	BLBS	R0, 12\$	
	67	64	50	E9	0007F	CLRB	@100(IRAB)	0733
	55		A7	9A	00082	BSBW	RMSGET_NEXT_KEY	0738
	55	10	A9	C0	00086	BLBC	R0, 14\$	
		64	65	94	0008A	MOVZBL	16(IDX_DFN), UPD_BUF_ADDR	0747
			5C	D4	0008C	ADDL2	100(IRAB), UPD_BUF_ADDR	
			A9	B1	0008E	CLRB	(UPD_BUF_ADDR)	0748
22	A7	56	0F	1F	00093	CLRL	AP	0750
		58	A9	DD	00095	CMPW	86(IRAB), 34(IDX_DFN)	0756
		0000G	04	C0	0009B	BLSSU	8\$	
	5E		50	E9	0009E	PUSHL	88(IRAB)	0758
	03		01	88	000A1	BSBW	RMSNULLKEY	
	65		52	B1	000A4	ADDL2	#4, SP	
22	A7		29	1F	000A8	BLBC	R0, 8\$	
			54	DD	000AA	BISB2	#1, (UPD_BUF_ADDR)	0760
		0000G	04	C0	000AF	CMPW	OLD_REC_SIZE, 34(IDX_DFN)	0764
	5E		50	E9	000B2	BLSSU	11\$	
	1E					PUSHL	OLD_REC_ADDR	0766
						BSBW	RMSNULLKEY	
						ADDL2	#4, SP	
						BLBC	R0, 11\$	

	13		65	E9	000B5	BLBC	(UPD_BUF_ADDR), 9\$	0773
			5C	D4	000B8	CLRL	AP	0781
	50	20	A7	9A	000BA	MOVZBL	32(IDX_DFN), R0	0788
	53	58	A9	D0	000BE	MOVL	88(IRAB), R3	
	51		54	D0	000C2	MOVL	OLD_REC_ADDR, R1	
			0000G	30	000C5	BSBW	RMSCOMPARE_KEY	
	05		50	E9	000C8	BLBC	R0, 10\$	
	65		02	88	000CB	BISB2	#2, (UPD_BUF_ADDR)	0792
			03	11	000CE	BRB	11\$	
	65		01	8A	000D0	BICB2	#1, (UPD_BUF_ADDR)	0799
			65	95	000D3	TSTB	(UPD_BUF_ADDR)	0805
			0C	13	000D5	BEQL	13\$	
07	1C	A7	01	E0	000D7	BBS	#1, 28(IDX_DFN), 13\$	0807
		50	8F	3C	000DC	MOVZWL	#33948, R0	0809
			2E	11	000E1	BRB	16\$	
	64	B9	65	88	000E3	BISB2	(UPD_BUF_ADDR), @100(IRAB)	0816
			93	11	000E7	BRB	7\$	0813
1B	64	B9	01	E1	000E9	BBC	#1, @100(IRAB), 15\$	0823
		03	CA	91	000EE	CMPB	183(IFAB), #3	0825
			14	1E	000F3	BGEQU	15\$	
7E	6C	A9	02	C1	000F5	ADDL3	#2, 108(IRAB), -(SP)	0832
			54	DD	000FA	PUSHL	OLD_REC_ADDR	
		7E	52	3C	000FC	MOVZWL	OLD_REC_SIZE, -(SP)	
			0000G	30	000FF	BSBW	RMSMOVE	
			0C	C0	00102	ADDL2	#12, SP	
	6C	B9	52	B0	00105	MOVW	OLD_REC_SIZE, @108(IRAB)	0833
			7E	D4	00109	CLRL	-(SP)	0838
			0000G	30	0010B	BSBW	RMSKEY_DESC	
		5E	04	C0	0010E	ADDL2	#4, SP	
			3C	BA	00111	POPR	#^M<R2,R3,R4,R5>	0840
			05	00	00113	RSB		

; Routine Size: 276 bytes, Routine Base: RMSRMS3 + 0123

; 779 0841 1

```
.. 781 0842 1 %SBTTL 'RMSUPDATE3B'
.. 782 0843 1 GLOBAL ROUTINE RMSUPDATE3B : RL$RABREG_67 =
.. 783 0844 1
.. 784 0845 1 ++
.. 785 0846 1
.. 786 0847 1 FUNCTIONAL DESCRIPTION:
.. 787 0848 1
.. 788 0849 1 High level update routine, using the following update algorithm:
.. 789 0850 1
.. 790 0851 1 1. If the operation is an $UPDATE (and not a converted $PUT), then the
.. 791 0852 1 current primary data record is retrieved using its RFA saved as part
.. 792 0853 1 of the next record context.
.. 793 0854 1
.. 794 0855 1 2. Record sizes, null key status, and comparisons between the new and
.. 795 0856 1 old versions of the current record are used to set flags in the
.. 796 0857 1 update buffer to determine which new secondary key values require
.. 797 0858 1 insertion and which old secondary key values require deletion. At
.. 798 0859 1 same time illegal key changes are detected.
.. 799 0860 1
.. 800 0861 1 3. If the file is a prologue 3 file, then the old version of the current
.. 801 0862 1 primary data record is saved in expanded format within the old record
.. 802 0863 1 buffer. If the file is a prologue 1 or 2 file, and there are old
.. 803 0864 1 secondary keys to be deleted, then the old version of the current
.. 804 0865 1 record will also be moved into the old record buffer.
.. 805 0866 1
.. 806 0867 1 4. New secondary key values are inserted. This involves:
.. 807 0868 1 a. Releasing the primary data bucket containing the current
.. 808 0869 1 primary data record.
.. 809 0870 1 b. Inserting the new secondary keys requiring insertion.
.. 810 0871 1 c. Reclaiming the primary data bucket containing the current
.. 811 0872 1 primary data record.
.. 812 0873 1
.. 813 0874 1 If the file is being RU ROLLBACK Recovered, then as the new SIDs
.. 814 0875 1 already exists in the file they are un-deleted instead of inserted.
.. 815 0876 1
.. 816 0877 1 5. The new version of the current record is inserted in place of the old
.. 817 0878 1 version, and any primary data bucket splits that are required in
.. 818 0879 1 order to make room for the new record are performed as is all
.. 819 0880 1 required updating of the primary index.
.. 820 0881 1
.. 821 0882 1 If the file is being RU Journalled and a Recovery Unit is in
.. 822 0883 1 progress, then the record is not allowed to shrink in size, and if
.. 823 0884 1 necessary even increases in size by one byte to reserve the space
.. 824 0885 1 the record currently occupies in case the Recovery Unit is aborted.
.. 825 0886 1
.. 826 0887 1 6. Old secondary key values requiring deletion are deleted. The keys
.. 827 0888 1 themselves are extracted from the old version of the current primary
.. 828 0889 1 data record which was saved earlier in the old record buffer.
.. 829 0890 1
.. 830 0891 1 If the file is being RU Journalled and a Recovery Unit is in
.. 831 0892 1 progress, then the old SIDs are just marked RU_DELETE and no space
.. 832 0893 1 is reclaimed.
.. 833 0894 1
.. 834 0895 1 CALLING SEQUENCE:
.. 835 0896 1
.. 836 0897 1 BSBW RMSUPDATE3B()
.. 837 0898 1
```

```
838 0899 1 INPUT PARAMETERS:
839 0900 1 NONE
840 0901 1
841 0902 1 IMPLICIT INPUTS:
842 0903 1
843 0904 1     IDX_DFN - primary key index descriptor
844 0905 1     - primary data bucket type
845 0906 1     - size of primary key
846 0907 1
847 0908 1     IFAB - address of IFAB
848 0909 1     IFBSB_PLG_VER - prologue version of file
849 0910 1     IFBSB_RFMORG - format of records
850 0911 1     IFBSV_RUP - if set, Recovery Unit in progress
851 0912 1     IFBSV_RU_RECVR - if set, RU ROLLBACK in progress
852 0913 1
853 0914 1     IRAB - address of IRAB
854 0915 1     IRBSV_ABOVELOCKD - if set, above level bucket is locked
855 0916 1     IRBSL_CURBDB - address of primary data bucket's BDB
856 0917 1     IRBSB_CUR_KREF - current NRP key of reference
857 0918 1     IRBSL_LOCK_BDB - address of above level bucket's BDB
858 0919 1     IRBSL_OLDBUF - address of old record buffer
859 0920 1     IRBSW_PUTUP_ID - ID of current primary data record
860 0921 1     IRBSL_PUTUPD_VBN - VBN of current primary data record
861 0922 1     IRBSW_RSZ - size of updated record
862 0923 1     IRBSV_UPDATE - if set, $UPDATE requested
863 0924 1     IRBSV_UPDATE_IF - if set, converted $PUT requested
864 0925 1     IRBSL_UPDBUF - address of update buffer
865 0926 1
866 0927 1     REC_ADDR - address of current primary data record
867 0928 1
868 0929 1 OUTPUT PARAMETERS:
869 0930 1 NONE
870 0931 1
871 0932 1 IMPLICIT OUTPUTS:
872 0933 1
873 0934 1     IRAB - address of IRAB
874 0935 1     IRBSB_CACHEFLGS - CSHSV_LOCK maybe set
875 0936 1     IRBSV_IDX_ERR - if set, error updating index
876 0937 1     IRBSW_LAST_ID - ID of record's last known position
877 0938 1     IRBSL_LAST_VBN - VBN of record's last known position
878 0939 1     IRBSL_LST_NCMP - last zero-front compressed key record
879 0940 1     IRBSW_SRCIFLAGS - IRBSV_POSINSERT maybe set
880 0941 1     IRBSB_STOPLEVEL - maybe set either to 0 or 1
881 0942 1
882 0943 1 ROUTINE VALUE:
883 0944 1
884 0945 1     See errors from:
885 0946 1
886 0947 1     RMSFIND_BY_RFA
887 0948 1     RMSFIND_BY_RRV
888 0949 1     RMSINS_ALL_SIDR
889 0950 1     RMSINSS_OR_IDX
890 0951 1     RMSPUT_OPD_CHKS
891 0952 1     RMSPUT_UPD_FIN
892 0953 1     RMSPUT_UPD_SPL
893 0954 1     RMSRLSBKT
894 0955 1     RMSRU_JOURNAL3
```



```

895 0956 1 | RMSUPDATE SCAN
896 0957 1 | RMSUPDEL COM
897 0958 1 |
898 0959 1 | SIDE EFFECTS:
899 0960 1 | NONE
900 0961 1 |
901 0962 1 | --
902 0963 1 |
903 0964 1 | BEGIN
904 0965 1 |
905 0966 1 | BUILTIN
906 0967 1 | AP,
907 0968 1 | TESTBITSS,
908 0969 1 | TESTBITCS;
909 0970 1 |
910 0971 1 | LABEL
911 0972 1 | GETBACK;
912 0973 1 |
913 0974 1 | EXTERNAL REGISTER
914 0975 1 | COMMON_RAB_STR,
915 0976 1 | R_IDX_DFN_STR,
916 0977 1 | R_REC_ADDR_STR;
917 0978 1 |
918 0979 1 | GLOBAL REGISTER
919 0980 1 | COMMON_IO_STR;
920 0981 1 |
921 0982 1 | ! This routine maybe called as the result of a $PUT converted into an
922 0983 1 | $UPDATE, or as the result of an $UPDATE. In the former case, this routine
923 0984 1 | would have been called with the primary data bucket locked, and RMS
924 0985 1 | positioned to the record to be updated so there would be no need to
925 0986 1 | perform this positioning. In the case of a straight $UPDATE, it would be
926 0987 1 | necessary to perform this positioning. To differentiate between the two
927 0988 1 | cases, the IRAB bit IRBSV_UPDATE is set on $UPDATES but cleared on
928 0989 1 | converted $PUTs on entry to this routine.
929 0990 1 |
930 0991 1 | On any errors from this point on, the routine RMSCLEAN_BDB will be called
931 0992 1 | to make sure any locked buckets, including the bucket which might have
932 0993 1 | been locked during lockabove optimization, are released.
933 0994 1 |
934 0995 1 | IF TESTBITSS (IRAB[IRBSV_UPDATE])
935 0996 1 | THEN
936 0997 1 | BEGIN
937 0998 1 |
938 0999 1 | ! This is an $UPDATE so make sure IRBSV_ABOVELOCKD is clear as there
939 1000 1 | cannot be any bucket locked at this time. Position to the record to
940 1001 1 | be updated.
941 1002 1 |
942 1003 1 | IRAB[IRBSV_ABOVELOCKD] = 0;
943 1004 1 |
944 1005 1 | RETURN_ON_ERROR (RMSUPDEL COM());
945 1006 1 | END;
946 1007 1 |
947 1008 1 | ! Perform some common checks and set up the primary key descriptor. These
948 1009 1 | checks have already been performed if this is a RU ROLLBACK Recovery
949 1010 1 | operation.
950 1011 1 |
951 1012 1 | IF NOT .IFAB[IFBSV_RU_RECVR]
```

```

952 1013 2 THEN
953 1014 RETURN_ON_ERROR (RM$PUT_UPD_CHKS(), RM$CLEAN_BDB());
954 1015
955 1016 ! Set up the update buffer indicating which alternate keys in the record to
956 1017 ! be updated will need inserting and which will need deleting.
957 1018
958 1019 RETURN_ON_ERROR (RM$UPDATE_SCAN(), RM$CLEAN_BDB());
959 1020
960 1021 IRAB[IRB$W_LAST_ID] = IRC$_ID(REC_ADDR);
961 1022
962 1023 ! Before any permanent modification to the file takes place, RU Journal this
963 1024 ! operation if RU Journalling is enabled and the process is within a
964 1025 ! Recovery Unit.
965 1026
966 1027 IF .IFAB[IFB$V_RUP]
967 1028 THEN
968 1029 BEGIN
969 1030
970 1031 LOCAL
971 1032 SAVE_REC_ADDR;
972 1033
973 1034 SAVE_REC_ADDR = .REC_ADDR;
974 1035 REC_ADDR = .IRAB[IRB$L_OLDBUF] + 2;
975 1036
976 1037 RETURN_ON_ERROR (RM$RU_JOURNAL3 (RJR$ UPDATE,
977 1038 .IRAB[IRB$L_PUTUP_VBN],
978 1039 .IRAB[IRB$W_PUTUP_ID],
979 1040 .(.IRAB[IRB$L_OLDBUF])<0,16>),
980 1041 RM$CLEAN_BDB());
981 1042
982 1043 REC_ADDR = .SAVE_REC_ADDR;
983 1044 END;
984 1045
985 1046 ! If any new alternate keys will have to be inserted as a result of this
986 1047 ! $UPDATE, then insert them at this time. To do so, first release the
987 1048 ! primary data bucket containing the old record, then insert all the
988 1049 ! alternate keys which need to be inserted, and finally re-position back to
989 1050 ! the old primary data record.
990 1051
991 1052 IF .BBLOCK[.IRAB[IRB$L_UPDBUF], UPD$V_INS_NEW]
992 1053 THEN
993 1054
994 1055 GETBACK:
995 1056 BEGIN
996 1057
997 1058 ! If RMS has performed lockabove optimization to lock the bucket from
998 1059 ! the level above, then release this bucket whose BDB has been saved in
999 1060 ! IRB$L_LOCK_BDB. This is because SDR inserts will also try to perform
1000 1061 ! lockabove optimization.
1001 1062
1002 1063 IF TESTBITSC (IRAB[IRB$V_ABOVELOCKD])
1003 1064 THEN
1004 1065 RELEASE (IRAB[IRB$L_LOCK_BDB]);
1005 1066
1006 1067 ! Release the primary data bucket containing the old record. Before the
1007 1068 ! bucket is released, the permanence bit is set so that RMS will
1008 1069 ! attempt to keep this bucket in the cache (it will eventually try to
```

```
1009      1070      3      ! re-position back to the old data record). The setting of the
1010      1071      3      ! permanence bit here should not ruin the setting of permanence for
1011      1072      3      ! alternate index root buckets with extra buffers because the bit
1012      1073      3      ! will be cleared when this bucket is reaccessed.
1013      1074      3
1014      1075      3      BDB = .IRAB[IRB$L_CURBDB];
1015      1076      3      IRAB[IRB$L_CURBDB] = 0;
1016      1077      3
1017      1078      3      IRAB[IRB$L_LAST_VBN] = .BDB[BDB$L_VBN];
1018      1079      3      BDB[BDB$V_PRM] = 1;
1019      1080      3      RMSRLSBKT(0);
1020      1081      3
1021      1082      3      ! If this is not a RU ROLLBACK Recovery operation then insert all the
1022      1083      3      ! alternate keys required by the new record.
1023      1084      3
1024      1085      3      IF NOT .IFAB[IFB$V_RU_RECVR]
1025      1086      3      THEN
1026      1087      4          RETURN_ON_ERROR (RMS$INS_ALL_SIDR())
1027      1088      4
1028      1089      4      ! If this is a RU ROLLBACK Recovery operation then it is not necessary
1029      1090      4      ! to insert any new alternate keys since they already exist in the file.
1030      1091      4      ! It is only necessary to position to and un-delete them.
1031      1092      4
1032      1093      3      ELSE
1033      1094      4          BEGIN
1034      1095      4
1035      1096      4          ! Signal that the SIDRs positioned to are to be un-deleted, and
1036      1097      4          ! position REC_ADDR to the new record so that the alternate keys
1037      1098      4          ! maybe extracted to be used in the positioning.
1038      1099      4
1039      1100      4          IRAB[IRB$V_RU_UNDEL] = 1;
1040      1101      4          REC_ADDR = .IRAB[IRB$L_RBF];
1041      1102      4
1042      1103      4          ! Continue until all the alternate keys of the new record have been
1043      1104      4          ! un-deleted.
1044      1105      4
1045      1106      4          WHILE RMS$GET_NEXT_KEY()
1046      1107      4          DO
1047      1108      4
1048      1109      4              ! If there is a new SIDR to be un-deleted for this key of
1049      1110      4              ! reference then do so.
1050      1111      4
1051      1112      4              IF .BBLOCK[.IRAB[IRB$L_UPDBUF] + .IDX_DFNC[IDX$B_DESC_NO],
1052      1113      4                  UPD$V_INS_NEW]
1053      1114      4              THEN
1054      1115      5                  BEGIN
1055      1116      5
1056      1117      5                  ! Extract the alternate key from the new record, and
1057      1118      5                  ! un-delete the corresponding SIDR.
1058      1119      5
1059      1120      5                  AP = 3;
1060      1121      5                  RMS$RECORD_KEY (KEYBUF_ADDR(2));
1061      1122      5                  RMS$DELETE_SIDR();
1062      1123      4                  END;
1063      1124      4
1064      1125      4          IRAB[IRB$V_RU_UNDEL] = 0;
1065      1126      3          END;
```



```
1066 1127 3
1067 1128 3
1068 1129 3
1069 1130 3
1070 1131 3
1071 1132 3
1072 1133 3
1073 1134 3
1074 1135 3
1075 1136 3
1076 1137 3
1077 1138 4
1078 1139 4
1079 1140 4
1080 1141 4
1081 1142 4
1082 1143 4
1083 1144 4
1084 1145 4
1085 1146 4
1086 1147 4
1087 1148 4
1088 1149 4
1089 1150 4
1090 1151 4
1091 1152 4
1092 1153 4
1093 1154 4
1094 1155 4
1095 1156 4
1096 1157 4
1097 1158 4
1098 1159 4
1099 1160 4
1100 1161 4
1101 1162 4
1102 1163 4
1103 1164 5
1104 1165 5
1105 1166 5
1106 1167 5
1107 1168 5
1108 1169 5
1109 1170 5
1110 1171 6
1111 1172 6
1112 1173 6
1113 1174 5
1114 1175 5
1115 1176 5
1116 1177 5
1117 1178 5
1118 1179 5
1119 1180 5
1120 1181 5
1121 1182 4
1122 1183 6

! Retrieve the primary key descriptor.
RMSKEY_DESC(0);

! Clear IRBSV_ABOVELOCKD as SISR inserts may have left this bit set.
IRAB[IRBSV_ABOVELOCKD] = 0;

! Position back to the current primary data record so it maybe changed.
BEGIN
LOCAL
STATUS;

! First, try to position back to the record using its last known
! location. This should succeed unless the bucket containing the
! current record has split, and the record has moved. The bit
! IRBSV_NORLS_RNF is set, so that RMS can always clear the BDB's
! permanence bit (set above before the bucket was released) regardless
! of whether it was successful at position to the current record or not.
IRAB[IRBSB_CACHEFLGS] = CSHM_LOCK;
IRAB[IRBSV_NORLS_RNF] = 1;

AP = .IRAB[IRBSW_LAST_ID];
STATUS = RMSFIND_BY_RFA (.IRAB[IRBSL_LAST_VBN]);

! If RMS was able to find the current record in its last know location
! verify, by comparing RFAs, that the record positioned to is in fact
! the current record. If so, clear the BDB's permanence bit and
! proceed with the update; otherwise, a second attempt will made to
! position to the current record utilizing its RFA.
IF .STATUS
THEN
BEGIN
AP = 3;
IF .IRAB[IRBSW_PUTUP_ID] EQL RMSRECORD_ID()
THEN
IF .IRAB[IRBSL_PUTUP_VBN] EQL RMSRECORD_VBN()
THEN
BEGIN
BBLOCK[.IRAB[IRBSL_CURBDB], BDBSV_PRM] = 0;
LEAVE GETBACK;
END;
END

! If RMS was unable to position to the current primary data record in
! its last known location because of some obscure I/O error, return
! that error; otherwise, a second attempt will be made to position to
! the current record utilizing its RFA.
ELSE
IF NOT (.STATUS<0,16> EQL RMSERR(RNF))
```

```
1123      1184      S
1124      1185      S
1125      1186      S
1126      1187      S
1127      1188      S
1128      1189      S
1129      1190      S
1130      1191      S
1131      1192      S
1132      1193      S
1133      1194      S
1134      1195      S
1135      1196      S
1136      1197      S
1137      1198      S
1138      1199      S
1139      1200      S
1140      1201      S
1141      1202      S
1142      1203      S
1143      1204      S
1144      1205      S
1145      1206      S
1146      1207      S
1147      1208      S
1148      1209      S
1149      1210      S
1150      1211      S
1151      1212      S
1152      1213      S
1153      1214      S
1154      1215      S
1155      1216      S
1156      1217      S
1157      1218      S
1158      1219      S
1159      1220      S
1160      1221      S
1161      1222      S
1162      1223      S
1163      1224      S
1164      1225      S
1165      1226      S
1166      1227      S
1167      1228      S
1168      1229      S
1169      1230      S
1170      1231      S
1171      1232      S
1172      1233      S
1173      1234      S
1174      1235      S
1175      1236      S
1176      1237      S
1177      1238      S
1178      1239      S
1179      1240      S

      .STATUS<0,16>or EQL RMSERR(DEL))
      THEN
      BEGIN
      RM$CLEAN BDB();
      RM$PUTUPD_ERROR();
      RETURN .STATUS;
      END;
      END;

      ! RMS was unable to locate the current primary data record in its
      ! last known location although it was able to access the primary data
      ! bucket in which it was last found. After clearing the permanence bit
      ! in the BDB for this primary data bucket and releasing the bucket, RMS
      ! attempts to position to the current primary data record by utilizing
      ! the RFA of the current record.
      BDB = .IRAB[IRB$L_CURBDB];
      BDB[BDB$V_PRM] = 0;
      RM$RLSBKT(0);
      IRAB[IRB$B_CACHEFLGS] = (SH$M_LOCK);

      RETURN ON ERROR
      (RM$FIND_BY_RRV (.IRAB[IRB$L_PUTUP_VBN], .IRAB[IRB$W_PUTUP_ID], 0),
      BEGIN
      RM$CLEAN BDB();
      RM$PUTUPD_ERROR();
      END);

      END;      ! of block GETBACK

      ! Having re-positioned to the current primary data record, RMS now replaces
      ! the old version of the record with the new version performing any primary
      ! data bucket splits and primary index updates that are required.
      BEGIN
      LOCAL
      STATUS,
      RECORD_SIZE,
      RECOVH_SIZE;

      ! Retrieve the address of the primary data bucket containing the current
      ! primary data record.
      BDB = .IRAB[IRB$L_CURBDB];
      BKT_ADDR = .BDB[BDB$L_ADDR];

      ! Determine the number of bytes the new version of the record will take up
      ! in the bucket, and the number of bytes the new version of the record will
      ! take up in the bucket including record overhead. Both of these quantities
      ! are prologue dependent, and the latter also depends upon the attributes
      ! of the records in the file.
      ! Also, reevaluate the record id for the record to be updated. It may
      ! easily have been displaced (as a result of a split, for example) since
      ! IRB$W_LAST_ID was last saved. We need this id in case the new record
      ! will be larger than the old one, causing a delete + insert operation
```

```
1180 1241 3 ! to be performed. If that happens, we need to use the same record id
1181 1242 3 ! for the newly inserted record to keep the same RFA.
1182 1243 3
1183 1244 3
1184 1245 3 IF .IFAB[IFBSB_PLG_VER] LSSU PLGSC_VER_3
1185 1246 3 THEN
1186 1247 3 BEGIN
1187 1248 3 RECORD_SIZE = .IRAB[IRBSW_RSZ];
1188 1249 3
1189 1250 3 RECOVH_SIZE = .RECORD_SIZE + IRCSC_FIXOVHDSZ;
1190 1251 3 IF .IFAB[IFBSB_RFMORG] NEQU FABSC_FIX
1191 1252 3 THEN
1192 1253 3 RECOVH_SIZE = .RECOVH_SIZE + IRCSC_DATSZFLD;
1193 1254 3 IRAB[IRBSW_LAST_ID] = .REC_ADDR[IRCSB_ID];
1194 1255 3 END
1195 1256 3 ELSE
1196 1257 3 BEGIN
1197 1258 3
1198 1259 3 ! The records in prologue 3 primary data buckets are packed. Record
1199 1260 3 ! packing includes key compression if it is enabled, and to insure that
1200 1261 3 ! the compression is done correctly, IRBSL_LST_NCMP is set to the last
1201 1262 3 ! known record with a zero front compressed primary key in the current
1202 1263 3 ! bucket.
1203 1264 3
1204 1265 3 IRAB[IRBSL_LST_NCMP] = .BKT_ADDR + BKTSC_OVERHDSZ;
1205 1266 3 RECORD_SIZE = RMSPACK_REC();
1206 1267 3
1207 1268 3 RECOVH_SIZE = .RECORD_SIZE + IRCSC_FIXOVHSZ3;
1208 1269 3 IF .IFAB[IFBSB_RFMORG] NEQU FABSC_FIX
1209 1270 3 OR
1210 1271 3 (.IFAB[IFBSB_RFMORG] EQLU FABSC_FIX
1211 1272 3 AND
1212 1273 3 .IDX_DFNC[IDXSB_DATBKTP] NEQU IDXSC_NCMPNCMP)
1213 1274 3 THEN
1214 1275 3 RECOVH_SIZE = .RECOVH_SIZE + IRCSC_DATSZFLD;
1215 1276 3 IRAB[IRBSW_LAST_ID] = .REC_ADDR[IRCSW_ID];
1216 1277 3 END;
1217 1278 3
1218 1279 3 ! Attempt to replace the old version of the current primary data record with
1219 1280 3 ! the new version without splitting the primary data bucket. The routine
1220 1281 3 ! RMSREPLACE will replace the old version with the new version if the latter
1221 1282 3 ! is the same size or smaller than the former. Failing that, it will delete
1222 1283 3 ! the old version of the primary data record, and RMSINSERT_UDR will try and
1223 1284 3 ! insert the new version into the primary data bucket.
1224 1285 3
1225 1286 3 IF NOT (STATUS = RMSREPLACE (.RECORD_SIZE))
1226 1287 3 THEN
1227 1288 3 BEGIN
1228 1289 3
1229 1290 3 ! If the size of the new record is smaller than the old record by one
1230 1291 3 ! byte and a specially formatted record must be built because the
1231 1292 3 ! process is currently within a Recovery Unit and the file is RU
1232 1293 3 ! Journalled, then increase the size of the record by two bytes to
1233 1294 3 ! allow for the specially formatted record to contain two record size
1234 1295 3 ! fields.
1235 1296 3
1236 1297 3 IF .IRAB[IRBSV_RU_UPDATE]
```



```
1237      1298  4      THEN
1238      1299  4          RECOVH_SIZE = .RECOVH_SIZE + IRCSC_DATSZFLD;
1239      1300  4
1240      1301  4      STATUS = RMSINSERT_UDR (RECOVH_SIZE);
1241      1302  4      END;
1242      1303  4
1243      1304  4      ! If RMS is successful at replacing the old version of the current primary
1244      1305  4      ! data record with the new version without causing a bucket split, then the
1245      1306  4      ! modified primary data bucket maybe marked dirty and released.
1246      1307  4
1247      1308  4      IF .STATUS
1248      1309  4      THEN
1249      1310  4          BEGIN
1250      1311  4              BDB[BDB$V_DRT] = 1;
1251      1312  4              IRAB[IRB$C_CURBDB] = 0;
1252      1313  4
1253      1314  4              RETURN_ON_ERROR (RMSRLSBKT (0),
1254      1315  4                  BEGIN
1255      1316  4                      RM$CLEAN_BDB();
1256      1317  4                      RM$PUTUPD_ERROR();
1257      1318  4                      END);
1258      1319  4
1259      1320  4              ! If an above level bucket was locked due to lockabove optimization,
1260      1321  4              ! then release it at this time.
1261      1322  4
1262      1323  4              IF (BDB = .IRAB[IRB$L_LOCK_BDB]) NEQU 0
1263      1324  4              THEN
1264      1325  5                  BEGIN
1265      1326  5                      RMSRLSBKT(0);
1266      1327  5                      IRAB[IRB$L_LOCK_BDB] = 0;
1267      1328  4                  END;
1268      1329  4              END
1269      1330  4
1270      1331  4              ! RMS was not able to replace the old version of the current primary
1271      1332  4              ! data record with the new version without splitting the primary data
1272      1333  4              ! bucket. Therefore, split the primary data bucket inserting the new
1273      1334  4              ! version of the current primary data record, and performing any index
1274      1335  4              ! updates that are required.
1275      1336  4
1276      1337  3      ELSE
1277      1338  4          BEGIN
1278      1339  4
1279      1340  4              ! Perform the split of the primary data level bucket and insert the new
1280      1341  4              ! version of the current primary data record.
1281      1342  4
1282      1343  4              IRAB[IRB$B_SPL_BITS] = 0;
1283      1344  4              IRAB[IRB$B_KEY$Z] = .IDX_DFN[IDX$B_KEYSZ];
1284      1345  4
1285      1346  4              RETURN_ON_ERROR (RM$PUT_UPD_SPL(.RECOVH_SIZE),
1286      1347  4                  BEGIN
1287      1348  4                      RM$CLEAN_BDB();
1288      1349  4                      RM$PUTUPD_ERROR();
1289      1350  4                      END);
1290      1351  4
1291      1352  4              ! Retrieve the BDB for the primary data bucket that has split, which is
1292      1353  4              ! still locked by this stream.
1293      1354  4
```

```
1294 1355 4 BDB = .IRAB[IRBSL_CURBDB];
1295 1356 4
1296 1357 4 ! If an index update is not required, just release the original primary
1297 1358 4 ! data bucket. The routine RMSPUT_UPD_SPL will clear the bit
1298 1359 4 ! IRBSV_UPDATE if an index update is not required following the primary
1299 1360 4 ! data bucket split, and set it if an index update is required.
1300 1361 4
1301 1362 4 IF TESTBITCS(IRAB[IRBSV_UPDATE])
1302 1363 4 THEN
1303 1364 4 BEGIN
1304 1365 4 IRAB[IRBSL_CURBDB] = 0;
1305 1366 4
1306 1367 4 RETURN_ON_ERROR (RMSRLSBKT(0), BEGIN
1307 1368 4 RMSCLEAN_BDB();
1308 1369 4 RMSPUTUPD_ERROR();
1309 1370 4 END);
1310 1371 4
1311 1372 4
1312 1373 4 ! If an above level bucket was locked due to lockabove optimization,
1313 1374 4 ! then release it at this time.
1314 1375 4
1315 1376 4 IF (BDB = .IRAB[IRBSL_LOCK_BDB]) NEQ 0
1316 1377 4 THEN
1317 1378 4 BEGIN
1318 1379 4 RMSRLSBKT(0);
1319 1380 4 IRAB[IRBSL_LOCK_BDB] = 0;
1320 1381 4 END;
1321 1382 4 END
1322 1383 4
1323 1384 4 ! The split of the primary data bucket requires updating of the primary
1324 1385 4 ! index. Perform the update at this time.
1325 1386 4
1326 1387 4 ELSE
1327 1388 4 BEGIN
1328 1389 4 IRAB[IRBSB_STOPLEVEL] = 1;
1329 1390 4 IRAB[IRBSW_SRCHFLAGS] = IRBSM_POSINSERT;
1330 1391 4
1331 1392 4 IF NOT (RAB[RABSL_STV] = RMSINSS_OR_IDX())
1332 1393 4 THEN
1333 1394 4 IRAB[IRBSV_IDX_ERR] = 1;
1334 1395 4
1335 1396 4 END
1336 1397 4 END;
1337 1398 4
1338 1399 4 END;
1339 1400 4
1340 1401 4 ! If there are alternate keys from the old record to be deleted, then the
1341 1402 4 ! old record has been saved in the IRBSL_OLDBUF record buffer. To delete
1342 1403 4 ! the old SIDs, the alternate key of each SIDR to be deleted is in turn
1343 1404 4 ! extracted into keybuffer 2 and the SIDR deletion routine is called.
1344 1405 4
1345 1406 4 IF .BBLOCK[.IRAB[IRBSL_UPDBUF],UPDSV_OLD_DEL]
1346 1407 4 THEN
1347 1408 4 BEGIN
1348 1409 4 REC_ADDR = .IRAB[IRBSL_OLDBUF] + 2;
1349 1410 4
1350 1411 4 ! If the current operation is an UPDATE_IF, then the current record may
```

```
1351 1412 1
1352 1413
1353 1414
1354 1415
1355 1416
1356 1417
1357 1418
1358 1419
1359 1420
1360 1421
1361 1422
1362 1423
1363 1424
1364 1425
1365 1426
1366 1427
1367 1428
1368 1429
1369 1430
1370 1431
1371 1432
1372 1433
1373 1434
1374 1435
1375 1436
1376 1437
1377 1438
1378 1439
1379 1440
1380 1441
1381 1442
1382 1443
1383 1444
1384 1445
1385 1446
1386 1447
1387 1448
1388 1449
1389 1450
1390 1451
1391 1452
1392 1453
1393 1454
1394 1455
1395 1456
1396 1457
1397 1458
1398 1459
1399 1460
1400 1461
1401 1462
1402 1463

! not be the record being updated. Yet, the routine RMSDELETE_SIDR
! operates only on the current record; therefore, setup the record
! being updated as the current record. Then is no need to restore these
! fields because they will be zeroed out at the conclusion of the
! operation anyway.
IF .IRAB[IRBSV_UPDATE_IF]
THEN
  BEGIN
    IRAB[IRBSL_UDR_VBN] = .IRAB[IRBSL_PUTUP_VBN];
    IRAB[IRBSW_UDR_ID] = .IRAB[IRBSW_PUTUP_ID];
  END;

! If this file is being Recovery Unit Journalled and the process is
! currently within a recovery unit the set the IRBSV_RU_DELETE state
! bit so that the SDRs being deleted are only marked RU_DELETE and no
! space is reclaimed.
IF .IFAB[IFBSV_RUP]
THEN
  IRAB[IRBSV_RU_DELETE] = 1;

! Loop through the index descriptors deleting those SDRs that need to
! be deleted.
WHILE RMSGET_NEXT_KEY()
DO
  ! If this index has a SDR which needs to be deleted, then delete
  ! it.
  IF .BBLOCK[.IRAB[IRBSL_UPDBUF] + .IDX_DFN[IDX$B_DESC_NO],
    UPDSV_OLD_DEL]
  THEN
    BEGIN
      ! Extract the alternate key from the unpacked version of the
      ! old data record, and delete the corresponding SDR.
      AP = 3;
      RMSRECORD_KEY (KEYBUF_ADDR(2));
      RMSDELETE_SIDR();
    END;

    IRAB[IRBSV_RU_DELETE] = 0;
  END;

! Having successfully completed the $UPDATE, finish up and return success.
RETURN RMSPUT_UPD_FIN()
END;
```


		34	BB	00000	RMSUPDATE3B::	
0A	04 06	5E	04	C2	00002	PUSHR #4M(R2,R4,R5> 0843
		A9	13	E3	00005	SUBL2 #4, SP 0995
		A9	20	8A	0000A	BBBS #19, 4(IRAB), 1\$ 1003
		4E	0000G	30	0000E	BICB2 #32, 6(IRAB) 1005
		06	50	E9	00011	BSBW RMSUPDDEL COM
		00A1	CA	E8	00014	BLBC STATUS, 6\$ 1012
		40	0000G	30	00019	BSBW 161(IFAB), 2\$ 1014
		3A	50	E9	0001C	BLBC RMSPUT_UPD_CHKS
		03	FECA	30	0001F	BSBW STATUS, 5\$ 1019
		50	00B7	CA	91	BLBC RMSUPDATE_SCAN
27	74 00A2	50	01	06	1E	CMPB STATUS, 5\$ 1021
		A9	01	A6	9A	BGEQU 183(IFAB), #3
		CA	50	04	11	3\$ 1(REC_ADDR), R0
		52	01	A6	3C	4\$ 1(REC_ADDR), R0
		A9	50	B0	00036	MOVW R0, 1T6(IRAB)
		7E	02	E1	0003A	BBC #2, 162(IFAB), 8\$ 1027
		7E	56	D0	00040	MOVL REC_ADDR, SAVE_REC_ADDR 1034
		0080	02	C1	00043	ADDL3 #2, 108(IRAB), -REC_ADDR 1035
		78	6C	B9	3C	MOVZWL @108(IRAB), -(SP) 1041
		0080	C9	3C	0004C	MOVZWL 128(IRAB), -(SP)
56	6C	7E	78	A9	DD	PUSHL 120(IRAB)
		05	1C	DD	00054	PUSHL #28
		05	0000G	30	00056	BSBW RMSRU JOURNAL3
		56	10	C0	00059	ADDL2 #16, SP
		03	50	E8	0005C	BLBS STATUS, 7\$
		03	0000G	30	0005F	BSBW RMSCLEAN_BDB
		03	43	11	00062	BRB 11\$
		03	52	D0	00064	MOVL SAVE_REC_ADDR, REC_ADDR 1043
		03	64	B9	E8	BLBS @100T(IRAB), 9\$ 1052
		03	00F7	31	0006B	BRW 18\$
11	04	A9	15	E5	0006E	BBCC #21, 4(IRAB), 10\$ 1063
		54	0084	C9	D0	MOVL 132(IRAB), BDB 1065
		54	0084	C9	D4	CLRL 132(IRAB)
		54	7E	D4	0007C	CLRL -(SP)
		54	0000G	30	0007E	BSBW RMSRLSBKT
		54	04	C0	00081	ADDL2 #4, SP
		54	20	A9	D0	MOVL 32(IRAB), BDB 1075
		54	20	A9	D4	CLRL 32(IRAB) 1076
		70	1C	A4	D0	MOVL 28(BDB), 112(IRAB) 1078
		0A	08	88	00090	BISB2 #8, 10(BDB) 1079
07	07	A9	7E	D4	00094	CLRL -(SP) 1080
		56	0000G	30	00096	BSBW RMSRLSBKT
		56	04	C0	00099	ADDL2 #4, SP
		56	00A1	CA	E8	BLBS 161(IFAB), 12\$ 1085
		56	0000G	30	000A1	BSBW RMSINS_ALL_SIDR 1087
		56	50	E8	000A4	BLBS STATUS, 15\$
		56	01DC	31	000A7	BRW 37\$
		56	40	8F	88	BISB2 #64, 7(IRAB) 1100
		56	58	A9	D0	MOVL 88(IRAB), REC_ADDR 1101
		56	0000G	30	000B3	BSBW RMSGET_NEXT_KEY 1106
07	07	22	50	E9	000B6	BLBC R0, 14\$
		50	10	A7	9A	MOVZBL 16(IDX_DFN), R0
		50	64	A9	C0	ADDL2 100(IRAB), R0
		EF	60	E9	000C1	BLBC (R0), 13\$ 1113

5C	03	D0	000C4	MOVL	#3, AP	1120
50	00B4	CA	3C 000C7	MOVZWL	180(IFAB), R0	1121
	60	B940	9F 000CC	PUSHAB	296(IRAB), R0	
		0000G	30 000D0	BSBW	RMSRECORD_KEY	
5E		04	C0 000D3	ADDL2	#4, SP	
		0000G	30 000D6	BSBW	RMSDELETE_SIDR	1122
		D8	11 000D9	BRB	13\$	1112
07	A9	40	8F 8A 000DB	BICB2	#64, 7(IRAB)	1125
		7E	D4 000E0	CLRL	-(SP)	1130
		0000G	30 000E2	BSBW	RMSKEY_DESC	
D6	A9	20	8A 000E5	BICB2	#32, 6(IRAB)	1134
40	A9	01	90 000E9	MOVB	#1, 64(IRAB)	1150
42	A9	20	88 000ED	BISB2	#32, 66(IRAB)	1151
	5C	74	A9 3C 000F1	MOVZWL	116(IRAB), AP	1153
	6E	70	A9 D0 000F5	MOVL	112(IRAB), (SP)	1154
		0000G	30 000F9	BSBW	RMSFIND_BY_RFA	
	5E	04	C0 000FC	ADDL2	#4, SP	
	52	50	D0 000FF	MOVL	R0, STATUS	
	22	52	E9 00102	BLBC	STATUS, 16\$	1162
	5C	03	D0 00105	MOVL	#3, AP	1166
		0000G	30 00108	BSBW	RMSRECORD_ID	1167
50		00	ED 0010B	CMPZV	#0, #16, T28(IRAB), R0	
	10	2D	12 00112	BNEQ	17\$	
		0000G	30 00114	BSBW	RMSRECORD_VBN	1169
	50	78	A9 D1 00117	CMPL	120(IRAB), R0	
		24	12 0011B	BNEQ	17\$	
	50	20	A9 D0 0011D	MOVL	32(IRAB), R0	1172
0A	A0		08 8A 00121	BICB2	#8, 10(R0)	
		3E	11 00125	BRB	18\$	1173
82B2	8F	52	B1 00127	CMPW	STATUS, #33458	1183
		13	13 0012C	BEQL	17\$	
8262	8F	52	B1 0012E	CMPW	STATUS, #33378	1185
		0C	13 00133	BEQL	17\$	
		0000G	30 00135	BSBW	RMSCLEAN_BDB	1188
		0000G	30 00138	BSBW	RMSPUTUPD_ERROR	1189
	50	52	D0 0013B	MOVL	STATUS, R0	1190
		0145	31 0013E	BRW	37\$	
	54	20	A9 D0 00141	MOVL	32(IRAB), BDB	1201
0A	A4		08 8A 00145	BICB2	#8, 10(BDB)	1202
		7E	D4 00149	CLRL	-(SP)	1203
		0000G	30 0014B	BSBW	RMSRLSBKT	
40	A9	01	90 0014E	MOVB	#1, 64(IRAB)	1204
		6E	D4 00152	CLRL	(SP)	1211
	7E	0080	C9 3C 00154	MOVZWL	128(IRAB), -(SP)	
		78	A9 DD 00159	PUSHL	120(IRAB)	
		0000G	30 0015C	BSBW	RMSFIND_BY_RRV	
	5E	0C	C0 0015F	ADDL2	#12, SP	
	7D	50	E9 00162	BLBC	STATUS, 27\$	
	54	20	A9 D0 00165	MOVL	32(IRAB), BDB	1229
	55	18	A4 D0 00169	MOVL	24(BDB), BKT_ADDR	1230
	03	00B7	CA 91 0016D	CMPB	183(IFAB), #3	1245
		18	1E 00172	BGEQU	20\$	
	50	56	A9 3C 00174	MOVZWL	86(IRAB), RECORD_SIZE	1248
	6E	07	A0 9E 00178	MOVAB	7(R0), RECOVH_SIZE	1250
	01	50	AA 91 0017C	CMPB	80(IFAB), #1	1251
		03	13 00180	BEQL	19\$	
	6E	02	C0 00182	ADDL2	#2, RECOVH_SIZE	1253

74	A9	01	A6	9B	00185	19\$:	MOVZBW	1(REC_ADDR), 116(IRAB)	1254
			21	11	0018A		BRB	23\$	1245
0098	C9	0E	A5	9E	0018C	20\$:	MOVAB	14(R5), 152(IRAB)	1265
			0000G	30	00192		BSBW	RMSPACK_REC	1266
	6E	09	A0	9E	00195		MOVAB	9(R0), RECOVH_SIZE	1268
	01	50	AA	91	00199		CMPB	80(IFAB), #1	1269
			06	12	0019D		BNEQ	21\$	
	06	29	A7	91	0019F		CMPB	41(IDX_DFN), #6	1273
			03	13	001A3		BEQL	22\$	
	6E		02	C0	001A5	21\$:	ADDL2	#2, RECOVH_SIZE	1275
74	A9	01	A6	B0	001AB	22\$:	MOVW	1(REC_ADDR), 116(IRAB)	1276
			50	DD	001AD	23\$:	PUSHL	RECORD_SIZE	1286
			FC17	30	001AF		BSBW	RMSREPLACE	
	5E		04	C0	001B2		ADDL2	#4, SP	
	13		50	E8	001B5		BLBS	STATUS, 25\$	
		07	A9	95	001B8		TSTB	7(IRAB)	1297
			03	18	001BB		BGEQ	24\$	
	6E		02	C0	001BD		ADDL2	#2, RECOVH_SIZE	1299
			5E	DD	001C0	24\$:	PUSHL	SP	1301
			0000G	30	001C2		BSBW	RMSINSERT_UDR	
	5E		04	C0	001C5		ADDL2	#4, SP	
	06		50	E9	001C8		BLBC	STATUS, 26\$	1308
0A	A4		02	88	001CB	25\$:	BISB2	#2, 10(BDB)	1311
			1D	11	001CF		BRB	28\$	1312
		44	A9	94	001D1	26\$:	CLRB	68(IRAB)	1343
00A6	C9	20	A7	90	001D4		MOVB	32(IDX_DFN), 166(IRAB)	1344
			6E	DD	001DA		PUSHL	RECOVH_SIZE	1350
			0000G	30	001DC		BSBW	RMSPUT_UPD_SPL	
	5E		04	C0	001DF		ADDL2	#4, SP	
	17		50	E9	001E2	27\$:	BLBC	STATUS, 29\$	
	54	20	A9	D0	001E5		MOVL	32(IRAB), BDB	1355
2C	04		13	E2	001E9		BBSS	#19, 4(IRAB), 31\$	1362
		20	A9	D4	001EE	28\$:	CLRL	32(IRAB)	1366
			7E	D4	001F1		CLRL	-(SP)	1371
			0000G	30	001F3		BSBW	RMSRLSBKT	
	5E		04	C0	001F6		ADDL2	#4, SP	
	09		50	E8	001F9		BLBS	STATUS, 30\$	
			0000G	30	001FC	29\$:	BSBW	RMSCLEAN_BDB	
			0000G	30	001FF		BSBW	RMSPUTUPD_ERROR	
			0081	31	00202		BRW	37\$	
	54	0084	C9	D0	00205	30\$:	MOVL	132(IRAB), BDB	1376
			24	13	0020A		BEQL	32\$	
			7E	D4	0020C		CLRL	-(SP)	1379
			0000G	30	0020E		BSBW	RMSRLSBKT	
	5E		04	C0	00211		ADDL2	#4, SP	
		0084	C9	D4	00214		CLRL	132(IRAB)	1380
			16	11	00218		BRB	32\$	1362
	41	A9	01	90	0021A	31\$:	MOVB	#1, 65(IRAB)	1390
	42	A9	01	B0	0021E		MOVW	#1, 66(IRAB)	1391
			0000G	30	00222		BSBW	RMSINSS_OR_IDX	1393
	0C	A8	50	D0	00225		MOVL	R0, 12(RAB)	
	04		50	E8	00229		BLBS	R0, 32\$	
	06	A9	02	88	0022C		BISB2	#2, 6(IRAB)	1395
	64	B9	01	E1	00230	32\$:	BBC	#1, 2100(IRAB), 36\$	1405
4E	6C	A9	02	C1	00235		ADDL3	#2, 108(IRAB), REC_ADDR	1409
56	06	A9	04	E1	0023A		BBC	#4, 6(IRAB), 33\$	1418
0D	00B0	C9	78	A9	D0	0023F	MOVL	120(IRAB), 176(IRAB)	1421

04	00BC 00A2 07	C9 CA A9	0080	C9 02 20	B0 E1 88	00245 0024C 00252	33\$:	MOVW	128(IRAB), 188(IRAB)	1422
				0000G	30	00256	34\$:	BBC	#2, 162(IFAB), 34\$	1430
		23		50	E9	00259		BISB2	#32, 7(IRAB)	1432
		50	10	A7	9A	0025C		BSBW	RMSGET_NEXT_KEY	1437
		50	64	A9	C0	00260		BLBC	R0, 35\$	1444
EE		60		01	E1	00264		MOVZBL	16(IDX_DFN), R0	
		5C		03	D0	00268		ADDL2	100(IRAB), R0	
		50	00B4	CA	3C	0026B		BBC	#1, (R0), 34\$	1451
			60	B940	9F	00270		MOVL	#3, AP	1452
				0000G	30	00274		MOVZWL	180(IFAB), R0	
		5E		04	C0	00277		PUSHAB	@96(IRAB)(R0)	
				0000G	30	0027A		BSBW	RMSRECORD_KEY	
				D7	11	0027D		ADDL2	#4, SP	1453
				0000G	30	0027F	35\$:	BSBW	RMSDELETE_SIDR	1443
07	A9			20	8A	0027F	36\$:	BRB	34\$	1456
				0000G	30	00283	37\$:	BICB2	#32, 7(IRAB)	1461
		5E		04	C0	00286		BSBW	RMSPUT_UPD_FIN	1463
				34	BA	00289		ADDL2	#4, SP	
				05	0028B			POPR	#^M<R2,R4,R5>	
								RSB		

: Routine Size: 652 bytes. Routine Base: RMSRMS3 + 0237

: 1403	1464	1
: 1404	1465	1 END
: 1405	1466	1
: 1406	1467	0 ELUDOM

PSECT SUMMARY

Name	Bytes	Attributes
RMSRMS3	1219	NOVEC,NOWRT, RD , EXE,NOSHR, GBL, REL, CON, PIC,ALIGN(2)

Library Statistics

File	Total	Symbols Loaded	Percent	Pages Mapped	Processing Time
_\$255\$DUA28:[RMS.OBJ]RMS.L32;1	3109	107	3	154	00:00.4

COMMAND QUALIFIERS

RM3UPDATE
V04-000

RMSUPDATE3B

E 14
16-Sep-1984 02:08:52
14-Sep-1984 13:01:43

VAX-11 Bliss-32 V4.0-742
[RMS.SRC]RM3UPDATE.B32;1

Page 35
(4)

; BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS\$:RM3UPDATE/OBJ=OBJ\$:RM3UPDATE MSRC\$:RM3UPDATE/UPDATE=(ENHS:RM3UPDATE)

; Size: 1219 code + 0 data bytes
; Run Time: 00:32.6
; Elapsed Time: 01:09.9
; Lines/CPU Min: 2702
; Lexemes/CPU-Min: 16349
; Memory Used: 286 pages
; Compilation Complete

**F

0328 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY